ERDC Major Shared Resource Center

ERDC MSRC/PET TR/00-01

Building Multidisciplinary Applications With MPI

by

Richard Weed

Work funded by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through

Programming Environment andTraining (PET)

Supported by Contract Number: DAHC94-96-C0002
Nichols Research Corporation

BUILDING MULTIDISCIPLINARY APPLICATIONS WITH MPI

Richard Weed *
Mississippi State University

November 2, 1999

1 INTRODUCTION

The requirements of higher accuracy and shorter simulation times in the modeling of coupled physical processes such as fluid-structure interaction, which requires the use of different solution algorithms in different parts of the global solution domain have lead researchers in the government, industry, and academia to pursue the development of "multidisciplinary applications." For the purposes of this report, a multidisciplinary application is defined as any application composed of two or more computer codes which perform different types of simulations or functions that are linked by some mechanism to allow the exchange of required data. For fluid-structure interactions, this could be a finite difference Computational Fluid Dynamics (CFD) code that provides pressure time-histories to a finite element structural dynamics code.

Prior to the introduction of parallel computing systems, coupling two different codes required that the developer either combine both codes into one code base and exchange data in memory or run the two codes sequentially and exchange data by I/O from mass storage. Both approaches are inherently serial and can be cumbersome to implement. However, accompanying the introduction of parallel computing hardware has been the development of software libraries that allow data messages to be passed among processes. Therefore, it is possible on parallel systems for the two codes to run concurrently as stand-alone processes and exchange data via the underlying communications hardware of the system. This approach allows the developer to minimize the amount of coding required to implement a coupled simulation and to run the simulations in parallel. All that is required is extra code to define the communication paths between different applications and to perform the actual sending and receiving of messages.

The purpose of this report is to describe a procedure for implementing multidisciplinary applications using the standard message passing software found on most parallel systems, the Message Passing Interface (MPI) library. In addition, it provides an introduction to a new suite of FORTRAN and C routines and Unix C shell (csh) scripts named MDARUN that implements the ideas discussed in this report. A detailed description of the software in the MDARUN package is given in an accompanying report [1]. The routines in the MDARUN package perform the main tasks required to implement a multidisciplinary application using MPI. The software is designed to reduce the effort required to implement an MPI-based multidisciplinary application.

2 SUMMARY OF THE IMPLEMENTATION PROCEDURE

The following discussion assumes the reader has some familiarity with MPI. Consult one of the several available references on MPI such as [2] if you have no prior experience with MPI programming. In addition,

*PET CSM Onsite Lead, ERDC MSRC, 1165 Porters Chapel Road, Vicksburg, MS 39180. E-mail: rweed@wes.hpc.mil

the ideas and procedures described in this report borrow heavily from the MPIRUN software package developed at NASA Ames Research Center by Fineberg [3]. The MPIRUN software package was used in prior research by the author [4] and has been utilized by U.S. Army Engineer Research and Development Center (ERDC) researchers developing a multidisciplinary application under the Department of Defense's Common High Performance Computing Software Initiative (CHSSI) program [5]. Unfortunately, the MPIRUN software is no longer actively supported and has proven to be difficult to maintain on the different types of parallel systems installed at the ERDC MSRC. Part of the motivation for this report and the software described in [1] was a need for software to replace the MPIRUN package on ERDC MSRC machines that can be easily maintained and implemented by individual developers.

One not so well-known feature of most (but, unfortunately, not all) MPI implementations is the ability to start groups of different (or the same) types of applications from a single loader command. The global communicator created by MPI, MPI_COMM_WORLD, will contain all of the applications started irrespective of the type or language of the different applications. This is illustrated in Figure 1. If two instances of a.out and two instances of b.out are started with the same loader command, the two a.out processes will have ranks 0 and 1 in MPI_COMM_WORLD and the two b.out processes will have ranks 2 and 3. For the simple example shown in Figure 1, communication between a.out and b.out can be accomplished using just MPI_COMM_WORLD. However, for a more complex application involving hundreds of processes and several different application groups, keeping up with who is who inside MPI_COMM_WORLD can be cumbersome. Therefore, the following procedure builds on the concept of groups of processes that is an integral part the MPI standard.

A "group" is defined in this report as one or more processes of the same application type. The group concept allows MPI_COMM_WORLD to be subdivided into smaller, more manageable units. Communication between these units is accomplished by building an "intercommunicator" between individual groups. In MPI syntax, intracommunicators are used for communications between members of the same group. Intercommunicators are used for communication between members of different groups. Using the group concept, building a multidisciplinary application can be boiled down to a three-step process. First, the application groups are defined a priori and unique intracommunicators are built for each group. Next, an intercommunicator is built on each member of individual groups that provides a path to the other application groups. Finally, communication between groups is accomplished by using the intercommunicator in the appropriate MPI send/receive routines. The details of each of these steps are presented in the following sections.

3 STEPS IN THE IMPLEMENTATION PROCESS

3.1 DEFINE THE APPLICATION GROUPS

Definition of the application groups requires the following of information: the number of different groups, the number of processes in each group (i.e., the number of copies of the application started for each group), and a unique index or tag that will be used to distinguish between different types of applications. In the MDARUN software described in [1], these data are defined either by the user hardwiring the appropriate information into their codes or by reading the information from a user-defined data file. In the MDARUN software, the initialization routine (*MDA_Init*) must be called by all members of MPI_COMM_WORLD. The process with rank 0 in MPI_COMM_WORLD is responsible for reading the group definition data or setting the hardwired values and then broadcasting that data to all processes in MPI_COMM_WORLD. With the number of processes in each group and type of application in each group defined, arrays containing the global rank in MPI_COMM_WORLD of the first process in each group (the group "leader") and a group number for each process in MPI_COMM_WORLD are created. At this point, all the data needed by individual processes to create a unique intracommunicator for each group will exist on all processes.

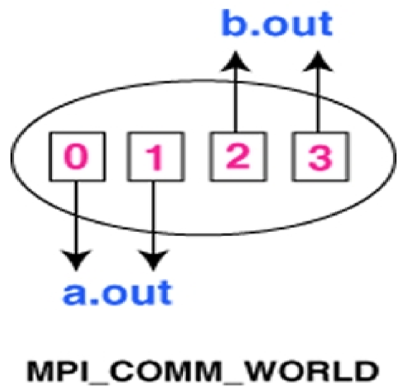

Figure 1: MPI_COMM_WORLD For Two Applications

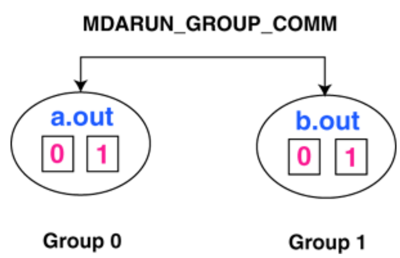

Figure 2: MDARUN_GROUP_COMM For Two Application Groups

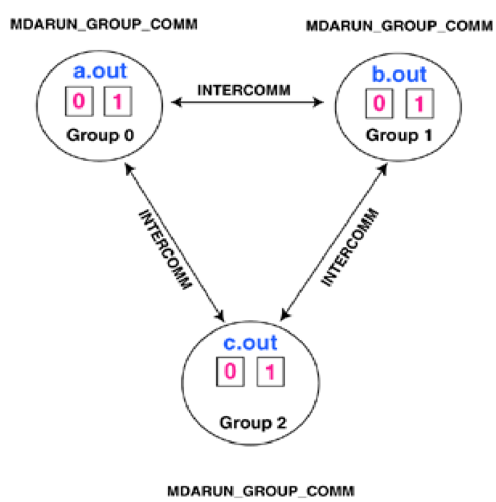

Figure 3: Intercommunicators Between Groups For Three Applications

The *MDA_Init* routine was designed to require a minimum amount of user input. The following lines are a sample data file for *MDA_Init* that defines four groups of two application types with each group containing two processes:

```
num_app_types  num_app_groups
2 4
nprocs/group  group_app_type
2 1
2 2
2 1
2 2
```

This example demonstrates the flexibility of the group concept. Different groups of the same application can work on different test cases or data sets. Consult [1] for more information on using *MDA_Init* and the data shown above.

3.2 CREATE THE GROUP INTRACOMMUNICATOR

The next step is to call the appropriate MPI function to create a group intracommunicator. This new communicator will only contain members of the same group. MPI provides two procedures for building group intracommunicators. The first procedure uses the MPI group and communicator creation functions. The steps in this procedure are given in the following pseudo-code:

```
On each process in MPI_COMM_WORLD,

  First, call MPI_Comm_group generate a global group from
  MPI_COMM_WORLD and then

      For each group,

        Define a list of the global ranks in MPI_COMM_WORLD
        of processes that will belong to each group

        Call MPI_Group_incl to create a new local group on
        each process using the global group and the group
        list

        Call MPI_Comm_create to create a temporary communicator for
        the each group

        If the current process is a member of the current group, then

          Set the group communicator equal to the temporary communicator

        Otherwise,

          Go to the next group

        End If

    End For
```

One problem with this approach is that all the processes must loop through the For loop in the above pseudo-code in the same order. At first glance, it would appear that each process should only have to execute one call to *MPI_Comm_create*. However, it is the author's experience that this procedure leads to inconsistent definitions for the communicators for each group. All processes must call *MPI_Comm_create* for all groups, even the groups which the current process is not a member. It is believed that this problem is a function of how MPI stores and book-keeps new communicators.

In the MDARUN software, a much simpler and less dangerous procedure was adopted. First, each process extracts its group number from the data defined in *MDA_Init*. This group number is then used as the "color" value in a call to the *MPI_Comm_split* function. *MPI_Comm_split* will create a new communicator containing all processes in MPI_COMM_WORLD with the same color value. In addition to being safer than the first approach described above, the new communicator is created with just one MPI function call.

The following FORTRAN code fragment illustrates how *MDA_Init* builds the MDARUN_GROUP_COMM communicator. The code extracts *my_group_number* and *my_local_rank* from the group definition data arrays and uses them as the color and "key" values in the *MPI_Comm_split* routine. The local rank is the rank of the current process in its group. The key value is used to fix the rank of the current process in the new communicator. The use of both the color and key values is shown in the following code fragment:

```
my_color = my_group_number
my_key = my_local_rank
Call MPI_Comm_split(MPI_COMM_WORLD, my_color, my_key, MDARUN_GROUP_COMM, ierr)
```

In the MDARUN software, this new group intracommunicator is called MDARUN_GROUP_COMM. Each process will have available a communicator named MDARUN_GROUP_COMM. However, each instance of MDARUN_GROUP_COMM is distinct to a unique group of processes. Each process in a group will have a local rank inside the MDARUN_GROUP_COMM communicator in the range of zero to the number of processes in the group minus one. The group leader is taken to be the process with local rank zero in MDARUN_GROUP_COMM. A communicator with the same name as the group communicator constructed by the MPIRUN package, MPIRUN_APP_COMM, is also constructed using the *MPI_Comm_dup* function. This communicator is provided by MDARUN to minimize the recoding required to move from the MPIRUN software to MDARUN.

The new group communicators are illustrated in Figure 2 for an application with two instances each of executables a.out and b.out.

3.3 CREATE THE GROUP INTERCOMMUNICATORS

The next step is to create an intercommunicator between the individual groups. This is accomplished by calling *MPI_Intercomm_create* using the unique group communicator for each group (MDARUN_GROUP_COMM or MPIRUN_APP_COMM), MPI_COMM_WORLD, and the global rank in MPI_COMM_WORLD of the leader or first process in the remote (other) group. For multiple groups this requires that each process perform multiple calls to *MPI_Intercomm_create* if intercommunication between some or all of the groups is desired. The following FORTRAN code fragment illustrates a typical call to *MPI_Intercomm_create*:

```
remote_rank = 20
Call MPI_Intercomm_create(MDARUN_GROUP_COMM, 0, MPI_COMM_WORLD,
                          remote_rank, tag, INTERCOMM, ierr)
```

In this example, the new group communicator, MDARUN_GROUP_COMM, will be used to create a new intercommunicator (INTERCOMM) with the group whose group leader has a global rank of 20 in MPI_COMM_WORLD.

As illustrated in the previous example, each call to *MPI_Intercomm_create* requires that the user supply the rank of the leader of the remote group in some "bridge" communicator which the processes in both the local and remote groups are a member. The bridge communicator is almost always taken to be MPI_COMM_WORLD. Therefore, the user must keep track of the global ranks inside MPI_COMM_WORLD of the group leaders. Although this information is generated by the *MDA_Init* routine, the user is forced to revert to thinking in terms of MPI_COMM_WORLD and the global ranks of processes.

The MDARUN software provides a wrapper routine, *MDA_Intergroup_comm*, that simplifies the creation of the intercommunicator by hiding the call to *MPI_Intercomm_create* from the user. *MDA_Intergroup_comm* requires only the remote group's group number and a "safe" tag as argument list input and returns the new communicator. The calculation of the appropriate global rank used in *MPI_Intercomm_create* is done inside *MDA_Intergroup_comm* and is hidden from the user. Using *MDA_Intergroup_comm* in place of *MPI_Intercomm_create* allows the user to define the intercommunicator in terms of the individual groups. Further details on using *MDA_Intergroup_comm* are given in [1].

The interaction of the group intercommunicators is shown in Figure 3. In each group, the process with local rank zero is the group leader.

4 USING THE GROUP INTERCOMMUNICATOR IN MPI FUNCTIONS

Once the intergroup communicator is defined, it can be used in MPI send and receive functions in place of the global MPI_COMM_WORLD communicator to transmit data to and from members of different groups. For example, the following call to the FORTRAN version of the *MPI_Isend* function can be used to send data from the current process to a process with local rank 2 in group 2. We will assume an intercommunicator called TO_GROUP_TWO was created by the procedure outlined in the previous sections.

```
Call MPI_Isend (sbuf, nbuf, MPI_DOUBLE_PRECISION, 2, mytag, TO_GROUP_TWO,
                ireq, ierr)
```

This is identical to an equivalent call to *MPI_Isend* using MPI_COMM_WORLD except for the use of the TO_GROUP_TWO communicator instead of MPI_COMM_WORLD and the local rank 2 instead of the equivalent global rank.

5 BUILDING INTERCOMMUNICATORS WITH MDARUN SOFTWARE

The following code fragments illustrate how easy it is to define the application groups and build the intercommunicators for a multidisciplinary application using the MDARUN software. The user adds the following calls to all applications:

FORTRAN:

```
  no_read=0
  md_unit=7
  Open(md_unit,file='my_data_file', STATUS='Unknown')
  Call MDA_Init(no_read, md_unit)
  .
  .
  .
  comm_tag=0
  Do n=0,mdarun_num_groups-1
    Call MDA_Intergroup_comm(n,comm_tag, INTERCOMM(n))
  End Do
```

C:

```
  no_read=0;
  md_unit=fopen("my_data_file","r");
  MDA_Init(no_read, md_unit);
  .
  .
  .
  comm_tag=0;
  for( n=0;n<=(mdarun_num_groups-1);n++ )
    MDA_Intergroup_comm(n,comm_tag, &INTERCOMM[n]);
```

This code will create an array of intercommunicators for all of the application groups. The local group communicator, MDARUN_GROUP_COMM, is returned when the group number n is the same as the group number of the local process. The variable *no_read* tells *MDA_Init* whether or not to read in the required initialization data. Setting *no_read* equal to one indicates that the required data will be defined by the user prior to the call to *MDA_Init*.

6 STARTING MULTIDISCIPLINARY JOBS ON THE ERDC MSRC PARALLEL SYSTEMS

The software and procedures described in this report for creating multidisciplinary applications have been tested on the ERDC MSRC SGI/CRAY Origin2000 and IBM SP systems. The commands used on these systems to start MPI jobs support loading multiple copies of different applications from the same command line. The commands for doing this are different on the two architectures. For example, on the SGI/CRAY Origin2000, the *mpirun* command can be used to start two instances each of applications a.out and b.out.

```
  mpirun -np 2 a.out : -np 2 b.out
```

A similar capability exists for the IBM *poe* command and its Portable Batch System (PBS) equivalent *pbspoe*. Details on using both *mpirun* and *poe*/*pbspoe* to start multidisciplinary applications and related issues are given in [1].

7 CHSSI MULTIDISCIPLINARY CODE IMPLEMENTATION USING MDARUN

This section describes how MDARUN is being used to support the CHSSI multidisciplinary code development described in [5]. The goal of the CHSSI EQM 1. project is to link two Computational Fluid Dynamics codes which use different solution algorithms and grid systems to solve the incompressible, Navier-Stokes equations for fluid flow. The first code, PAR3D, uses a finite volume discretization on structured hexahedral grid systems. The second code, GLS3D, uses a Galerkin Least Squares finite element solution technique on unstructured tetrahedral meshes. PAR3D is written in FORTRAN 77 and GLS3D is written in C. The multidisciplinary application being developed from these two codes will utilize the strengths of both codes to provide a coupled simulation capability for hydraulic flows in rivers or around ocean coastlines where the boundaries of the computational domains may be highly jagged and irregular. The unstructured tetrahedral mesh system is more capable of handling the irregular mesh topology. On the other hand, finite volume codes such as PAR3D are more cost effective and, in general, provide better local conservation of flow quantities than finite element codes. By coupling the two codes, the strengths of both solution techniques can be utilized.

The initial implementation of the coupled PAR3D/GLS3D application used the MPIRUN package described in [3]. The MPIRUN software was used to form an intercommunicator between the two codes. This enabled the two codes to exchange flow quantities such as velocities and pressures along the interfaces of the computational domains of each code. As stated in previous sections of this report, the MPIRUN software has proven difficult to maintain across the different parallel system architectures at the ERDC MSRC. The CHSSI code developers, Drs. Robert Bernard and Charlie Berger of the ERDC Coastal Hydraulics Laboratory, needed a software package that they could implement directly into their codes that was easily maintained and transportable across several systems. These requirements motivated the development of the MDARUN package.

The MDARUN package has been used to replace the MPIRUN software and data in the PAR3D code. Implementation into the GLS3D code is underway. In PAR3D, the modifications consisted of replacing the call to *MPIRUN_INIT* with a call to *MDA_Init* and replacing the MPIRUN_APP_COMM communicator and MPIRUN_APP_LEADERS array with their MDARUN equivalents. The modified PAR3D code was tested by starting two groups of PAR3D applications and comparing the results with the MPIRUN version of the code. The MDARUN version of PAR3D ran without incident on the ERDC MSRC Origin2000 system.

8 CONCLUDING REMARKS

This report has outlined a procedure for using MPI to create multidisciplinary applications. In addition, the main routines of a small software package called MDARUN which is designed to reduce the amount of work required to implement a multidisciplinary application from legacy codes were introduced. The new software builds on the MPI group concept and provides simple interfaces to the underlying MPI routines required to build multidisciplinary applications. It is hoped that the information provided in this report and the MDARUN software described in the accompanying report will enable ERDC MSRC users developing multidisciplinary applications to reduce the amount of time required to develop their applications.

References

[1] Weed, R., "MDARUN - A Package of Software for Creating Multidisciplinary Applications with MPI," ERDC MSRC PET TR-00-02, August, 1999.

[2] Snir, M., et al., *MPI - The Complete Reference*, MIT Press, 1996.

[3] Fineberg, S., "Implementing Multi-disciplinary and Multi-zonal Applications Using MPI," Report No. NAS-95-003, NASA Ames Research Center, January, 1995.

[4] Goodwin, S., Weed, R., Sankar, L., and Raj, P., "Towards Cost-Effective Aeroelastic Analysis on Advanced Parallel Computing Systems," AIAA Paper No. 97-0646, Reno, Nevada, January 6-10, 1997.

[5] Bernard, R., "Structured-Unstructured Modeling," CHSSI EQM 1, http://www.hpcmo.hpc.mil/Htdocs/CTAs/index.html, 1999.